(19) JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11) Publication number: 09062639 A

(43) Date of publication of application: 07.03.97

(51) Int. Cl G06F 15/163

(21) Application number: 07215601

(22) Date of filing: 24.08.95

(71) Applicant: IBM JAPAN LTD

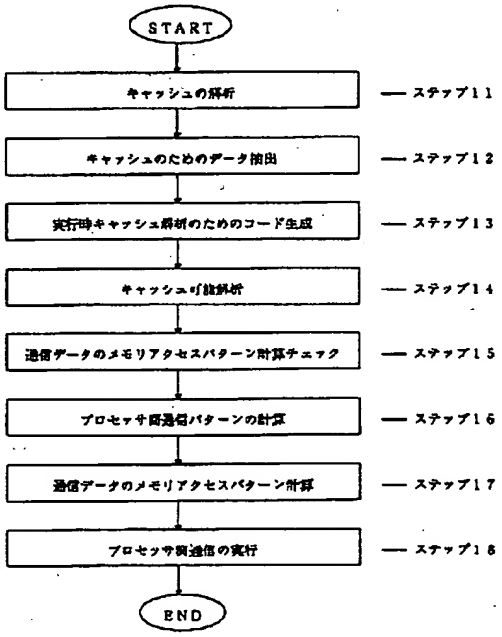
(72) Inventor: ISHIZAKI KAZUAKI
KOMATSU HIDEAKI
OGASAWARA TAKESHI

(54) INTER-PROCESSOR COMMUNICATION METHOD OF PARALLEL COMPUTER

(57) Abstract:

PROBLEM TO BE SOLVED: To speed up a communication between processors by comparing parameters based upon the execution of an execution-time code with parameters stored in a work area, and making the processors communicate with each other by reusing a communication pattern or memory access pattern according to the comparison result.

SOLUTION: It is analyzed whether or not a communication pattern when a loop nest is executed can be cached (step 11) and a plurality of parameters determining the communication pattern between the processors is extracted (steps 12 and 13). Then the current parameters in the work area and old parameters stored in the work area are compared with each other (step 14), it is checked whether or not cached data are the same according to the comparison result (step 15), and the communication pattern or memory access pattern is reused to carry out the communication between the processors (steps 16-18). Consequently, the communication between the processors can be speeded up.



Japanese Patent Laid-open No. Hei 9-62639

[Abstract] (with modification)

[Problem] To carry out high-speed communication between processors by using communication history in the past and omitting calculation of a communication pattern.

[Constitution] In a parallel computer, there are a step of extracting a plurality of parameters determining a communication pattern between processors, a step of preparing a work area for storing the parameters, a step of generating an execution-time code for storing the parameters in the work area, a step of storing the parameters in the work area for the purpose of storing the history of the communication pattern or the history of a memory access pattern for reading and writing data for communication between the processors, and a step of, in the case where the execution-time code is executed, comparing the parameters based on the execution of the execution-time code with the parameters stored in the work area. According to the comparison result, the communication pattern or the memory access pattern is reused to execute communication between the processors.

[Scope of Claims for Patent]

[Claim 1] A method of executing communication between processors in a computer having a plurality of processors characterized by comprising the steps of:

(a) extracting a plurality of parameters determining a communication pattern between said processors;

(b) preparing a work area for storing said parameters;

(c) generating an execution-time code for storing said parameters in said work area;

(d) storing said parameters in said work area for the purpose of storing the history of said communication pattern or the history of a memory access pattern for reading and writing data for communication between said processors;

(e) in the case where said execution-time code is executed, comparing said parameters based on the execution of said execution-time code with said parameters stored in said work area; and

(f) according to the comparison result, reusing said communication pattern or said memory access pattern to execute communication between said processors.

[Claim 2] A method as claimed in claim 1, characterized in that extraction of said parameters in the above step (a) is carried out by checking whether the values remain constant or change during execution of a loop nest and by judging whether

said communication pattern is chacheable or not.

[Claim 3] A method as claimed in claim 1, characterized in that the above step (e) is comparing said parameters based on the execution of said execution-time code with said parameters with regard to said communication pattern stored in said work area to check whether they are the same.

[Claim 4] A method as claimed in claim 1, characterized in that the above step (e) is comparing said parameters based on the execution of said execution-time code with said parameters with regard to said communication pattern stored in said work area to check whether they are the same, and, in the case where they are the same, comparing said parameters based on the execution of said execution-time code with said parameters with regard to said memory access pattern to check whether they are the same.

[Claim 5] A method as claimed in claim 1 or 2, characterized in that said parameters extracted in the above step (a) have parameters in an equation for deciding parameters with regard to a loop in a program, the number of processors for dividing the arrays on the left side and the right side in an assignment statement in said loop, the size of the respective dimensions of said arrays, the method of dividing said respective dimensions of said arrays, and reference of said

arrays.

[Claim 6] A method as claimed in claim 1, characterized in that said communication pattern describes an area for transferring data from one processor to another processor.

[Claim 7] A method as claimed in claim 1, characterized in that said memory access pattern describes a memory area managed by a processor when data is transferred according to said communication pattern.

[Claim 8] A method as claimed in claim 1, characterized in that the above step (f) uses said parameters stored in said work area.

[Claim 9] A method of executing communication between processors in a computer having a plurality of processors characterized by comprising the steps of:

(a) extracting a plurality of parameters determining a communication pattern between said processors;

(b) preparing a work area for storing said parameters for the purpose of storing the history of said communication pattern or the history of a memory access pattern; and

(c) generating an execution-time code for storing said parameters in said work area.

[Claim 10] A method of executing communication between processors in a computer having a plurality of processors

characterized by comprising the steps of:

(a) storing in a work area secured in advance a plurality of parameters determining a communication pattern between said processors;

(b) in the case where an execution-time code is executed, comparing parameters based on the execution of said execution-time code with said parameters stored in said work area; and

(c) according to the comparison result, reusing said communication pattern or said memory access pattern.

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平９－６２６３９

(43)公開日　平成９年（１９９７）３月７日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
G06F 15/163			G06F 15/16	320 K	

審査請求　未請求　請求項の数１０　ＯＬ　（全１２頁）

(21)出願番号　特願平７－２１５６０１

(22)出願日　平成７年（１９９５）８月２４日

(71)出願人　５９２０７３１０１
日本アイ・ビー・エム株式会社
東京都港区六本木３丁目２番１２号

(72)発明者　石崎　一明
神奈川県大和市下鶴間１６２３番地１４
日本アイ・ビー・エム株式会社東京基礎研究所内

(72)発明者　小松　秀昭
神奈川県大和市下鶴間１６２３番地１４
日本アイ・ビー・エム株式会社東京基礎研究所内

(74)代理人　弁理士　合田　潔　（外２名）

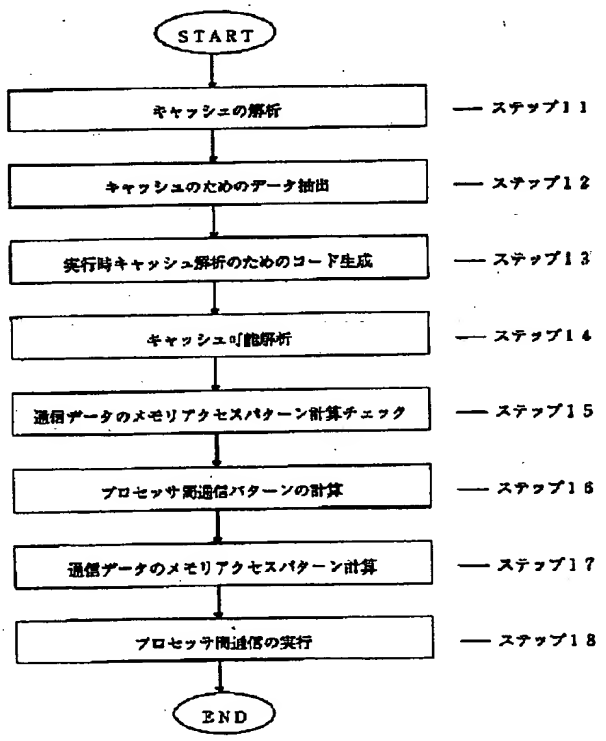
最終頁に続く

(54)【発明の名称】並列計算機のプロセッサ間通信方法

(57)【要約】　（修正有）

【課題】過去の通信履歴を用いて通信パターンの計算を省略し、高速にプロセッサ間通信を行う。

【解決手段】並列計算機において、プロセッサ間の通信パターンを決定する複数のパラメータを抽出するステップ、前記パラメータを保存する作業領域を作成するステップ、前記パラメータを作業領域に格納する実行時コードを生成するステップ、通信パターンの履歴又はプロセッサ間の通信のためのデータの読み書きを行うメモリアクセスパターンの履歴を記憶するために、作業領域に前記パラメータを格納するステップ、及び実行時コードが実行された場合、実行時コードの実行に基づいた前記パラメータを作業領域に記憶された前記パラメータと比較するステップを有し、その比較結果に応じて、通信パターン又はメモリアクセスパターンを再利用して、プロセッサ間の通信を実行する。



1

【特許請求の範囲】

【請求項１】複数のプロセッサを有するコンピュータにおけるプロセッサ間の通信を実行する方法において、(a) プロセッサ間の通信パターンを決定する複数のパラメータを抽出するステップと、(b) 前記パラメータを保存する作業領域を作成するステップと、(c) 前記パラメータを前記作業領域に格納する実行時コードを生成するステップと、(d) 通信パターンの履歴またはプロセッサ間の通信のためのデータの読み書きを行うメモリアクセスパターンの履歴を記憶するために、前記作業領域に前記パラメータを格納するステップと、(e) 前記実行時コードが実行された場合、前記実行時コードの実行に基づいた前記パラメータを前記作業領域に記憶された前記パラメータと比較するステップと、(f) その比較結果に応じて、前記通信パターン又は前記メモリアクセスパターンを再利用して、プロセッサ間の通信を実行するステップとを有することを特徴とする方法。

【請求項２】上記ステップ(a)における前記パラメータの抽出は、定数又はループネストを実行している間に値が変化するかどうかを調べ、前記通信パターンがキャッシュ可能であるかどうかを判断することにより行うことを特徴とする請求項１に記載の方法。

【請求項３】上記ステップ(e)は、前記実行時コードの実行に基づいた前記パラメータが、前記作業領域に記憶された前記通信パターンに関する前記パラメータと一致するかどうかを比較することを特徴とする請求項１に記載の方法。

【請求項４】上記ステップ(e)は、前記実行時コードの実行に基づいた前記パラメータが、前記作業領域に記憶された前記通信パターンに関する前記パラメータと一致するかどうかを比較するとともに、前記パラメータと一致する場合には、さらに前記メモリアクセスパターンに関する前記パラメータと一致するかどうかを比較することを特徴とする請求項１に記載の方法。

【請求項５】上記ステップ(a)において抽出される前記パラメータは、プログラム中のループに関するパラメータ、前記ループ内の代入文中の左辺及び右辺の配列を分割するプロセッサの個数、前記配列の各次元の大きさ、前記配列の各次元の分割方法、前記配列参照を決定する式にパラメータを有することを特徴とする請求１又は２に記載の方法。

【請求項６】前記通信パターンは、一のプロセッサから他のプロセッサへデータを受け渡しをする領域を記述したものであることを特徴とする請求項１に記載の方法。

【請求項７】前記メモリアクセスパターンは、前記通信パターンに従いデータを受け渡しする際に、あるプロセッサが管理するメモリ領域を記述したものであることを特徴とする請求項１に記載の方法。

【請求項８】上記ステップ(f)は、前記作業領域に格納された前記パラメータを用いていることを特徴とする請求項１に記載の方法。

2

【請求項９】複数のプロセッサを有するコンピュータにおけるプロセッサ間の通信を実行する方法において、(a) プロセッサ間の通信パターンを決定する複数のパラメータを抽出するステップと、(b) 通信パターンの履歴またはメモリアクセスパターンの履歴を記憶するために、前記パラメータを保存する作業領域を作成するステップと、(c) 前記パラメータを前記作業領域に格納する実行時コードを生成するステップとを有することを特徴とする方法。

【請求項１０】複数のプロセッサを有するコンピュータにおけるプロセッサ間の通信を実行する方法において、(a) プロセッサ間の通信パターンを決定する複数のパラメータを、予め確保された作業領域に格納するステップと、(b) 実行時コードが実行された場合、前記実行時コードの実行に基づいたパラメータを前記作業領域に記憶された前記パラメータと比較するステップと、(c) その比較結果に応じて、前記通信パターン又は前記メモリアクセスパターンを再利用するステップとを有することを特徴とする方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、並列計算機のプロセッサ間通信方法に係り、特に過去の通信履歴を用いてプロセッサ間の通信パターンの計算を省略し、高速にプロセッサ間通信を実行する方法に関する。

【０００２】

【従来の技術】複数のプロセッサを有し、それぞれのプロセッサがメモリを有するコンピュータシステムにおいて、プログラムを高速に実行するためには、データをそれぞれのプロセッサに分割し、データに対する演算をそれぞれのプロセッサに分割する必要がある。この場合、プロセッサ間でデータを受け渡すためにプロセッサ間でデータの受け渡し、すなわち通信が発生する。マルチプロセッサシステムでプログラムを高速に実行するためには、この通信を高速化することが重要である。

【０００３】プロセッサ間の通信を高速化する方法として、以下のような従来技術が知られている。配列ｘの要素ベクタｋが与えられたとき、その要素をもつプロセッサを与える関数を$D_j(k)$とする。また、ループネストｊにおいて、ループインデックスベクタｉが与えられたとき、ループインデックスベクタｉの計算を実行するプロセッサを与える関数を$C_j(i)$とする。さらにループネストｊにおいて、ループインデックスベクタｉが与えられたとき、ｊで参照する配列要素ｘを決定する関数を$F_j(i)$とする。このとき、次式が成立するならば、プロセッサ間の通信は発生しないが、成立しないならば通信が発生する。

【０００４】

【数１】$D_j(F_j(i)) = C_j(i)$

【０００５】この式は、ｉというインデックスでアクセスされる配列ｘの要素をもつプロセッサとｉというインデックスの計算を行うプロセッサが等しいことを示している。

【０００６】通信が発生する場合、$D_x(F_{j1}(i))$からC_j(i)へ転送を行うプロセッサ間の通信パターンを決定する必要がある。

【０００７】以下では、プログラム中の代入文における通信の説明を簡単にするために、計算を行うプロセッサを求める関数$C_j(i)$を、代入文の左辺の配列の分割と同一にする。この場合、プロセッサ間の通信が必要であるかどうかを求めるには、代入文の左辺の配列ＬＨＳ(Left-Hand-Side)と右辺の配列ＲＨＳ(Right-Hand-Side)について、以下の式が成立するかどうかを調べればよい。

【０００８】

【数２】

$$D_{LHS}(F_{j,LHS}(i)) = C_{RHS_j}(F_{j,RHS}(i))$$

【０００９】この式は、ｉというインデックスでアクセスされる配列ＬＨＳの要素をもつプロセッサと、ｉというインデックスでアクセスされる配列ＲＨＳの要素をもつプロセッサが等しいことを示している。

【００１０】代入文でアクセスされる配列の分割は、配列を分割するプロセッサ数、各次元の大きさ及び分割方法によって決定される。ここで、以下のようなベクタを定義する。

【００１１】ベクタｐ：配列の各次元を分割するプロセッサの個数を示すベクタ
ベクタｎ：配列の各次元の大きさを示すベクタ
ベクタｄ：配列の各次元の分割方法を示すベクタ
ベクタａ：配列参照を決定する式（ループインデックス変数に関する一次式）のパラメータ

【００１２】このような定義において、配列ｘの分割を決定する関数$D_x(k)$は、ベクタｐ，ベクタｎ及びベクタｄで決定される。また、ループインデックスベクタは、ループインデックスの上下限によって決定される。さらに、ループネストｊで、ループインデックスベクタｉが与えられたとき、参照する配列要素を決定する関数$F_{j1}(i)$はベクタａで決定される。

【００１３】従って、ある代入文を実行する際に必要となる通信を決定するには、プログラム中の以下のパラメータが必要になる。

【００１４】(1) ループインデックスの上下限
(2) 左辺の配列の各次元を分割するプロセッサ個数
(3) 左辺配列の各次元の大きさ
(4) 左辺の配列の各次元の分割方法
(5) 左辺の配列参照を決定する式のパラメータ
(6) 右辺の配列の各次元を分割するプロセッサ個数
(7) 右辺配列の各次元の大きさ
(8) 右辺の配列の各次元の分割方法
(9) 右辺の配列参照を決定する式のパラメータ

【００１５】これを以下のプログラムリスト１を例に説明する。なお、一般に与えられたプログラムでは、アクセスされる配列の内容は異なっているが、プロセッサ間のデータ通信パターンは繰り返し実行される部分は同じであることが多いことが知られている。

【００１６】

【数３】

［プログラムリスト１］

```
      SUBROUTINE SUB(A,N)
      REAL A(N)
!HPF$ PROCESSORS P(4)
!HPF$ DISTRIBUTE (BLOCK) onto P :: A
      DO TIME=1,10
        DO I=2,N-1
          A(I)=A(I-1)+A(I)+A(I+1)
        ENDDO
      ENDDO
      END
```

【００１７】なお、この例で、「!HPF$ PROCESSORS P(4)」というステートメントは、４つのプロセッサを１次元に配置し、そのプロセッサ配置にＰという名前を付けることを意味する。また、「!HPF$ DISTRIBUTE (BLOCK) onto P :: A」というステートメントは、配列Ａの分割方法を表し、具体的には、配列Ａを、ブロック分割し、Ｐというプロセッサに配置することを意味する。また、「A(I)=A(I-1)+A(I)+A(I+1)」を代入文という。

【００１８】プログラム１において、上述のパラメータは、以下のようになる。

【数４】

(1) ループインデックスの上下限：　2,N-1
(2) 左辺の配列の各次元を分割するプロセッサ個数：1次元目 4
(3) 左辺配列の各次元の大きさ：　1次元目 N
(4) 左辺の配列の各次元の分割方法：　1次元目 BLOCK
(5) 左辺の配列参照を決定する式のﾊﾟﾗﾒｰﾀ(aI+b)：　1次元目 a=1,b=0
(6) 右辺の配列の各次元を分割するプロセッサ個数：1次元目 4
(7) 右辺配列の各次元の大きさ：　1次元目 N
(8) 右辺の配列の各次元の分割方法：　1次元目 BLOCK
(9) 右辺の配列参照を決定する式のﾊﾟﾗﾒｰﾀ(aI+b)：　1次元目 a=1,b=-1

【００１９】従来の方法では、DO Ｉのループネストが実行されるたびに繰り返し通信パターンを計算し、計算された通信パターンに従って、相手プロセッサにデータを送信していた。従って、通信が発生する度に、通信パターンを計算し、プロセッサ間通信を行う時間が必要であった。

【００２０】

【発明が解決しようとする課題】上述のように、従来の方法では、プロセッサ間のデータの受け渡しに必要となる度に、通信パターンを計算する必要が生じていた。この通信パターンの計算に要する時間が、通信におけるオーバーヘッドとなっていた。そこで、本発明は、プロセッサ間の通信を高速化することを目的とする。

【００２１】

【課題を解決するための手段】上記のプログラムリスト１において、DO Ｉのループネストが繰り返し実行される度に、上記のパラメータは変更がない。ループネストが実行される際の通信パターンは、TIME=1の時に一度計算してしまえば、TIME=2...10の次回以降は通信パターンの計算を行わずに、既に計算された通信パターンを再利用してデータを送信すればよく、プロセッサ間の通信パターンの計算を省略することができる。

【００２２】そこで本発明は、複数のプロセッサを有するコンピュータにおけるプロセッサ間の通信を実行する方法において、(a) プロセッサ間の通信パターンを決定する複数のパラメータを抽出するステップと、(b) パラメータを保存する作業領域を作成するステップと、(c) パラメータを作業領域に格納する実行時コードを生成するステップと、(d) 通信パターンの履歴、またはプロセッサ間の通信のためのデータの読み書きを行うメモリアクセスパターンの履歴を記憶するために、作業領域にパラメータを格納するステップと、(e) 実行時コードが実行された場合、実行時コードの実行に基づいたパラメータを作業領域に記憶されたパラメータと比較するステップと、(f) その比較結果に応じて、作業領域に格納されたパラメータを使用することにより、通信パターン又はメモリアクセスパターンを再利用して、プロセッサ間の通信を実行するステップとを有する方法を提供する。

【００２３】ここで、ステップ(a)は、それぞれのパラメータが、ループネストを実行している間に値が変化するか、定数かどうかを調べることにより、通信パターンがキャッシュ可能であるかどうかを判断することにより抽出されるようにしてもよい。

【００２４】また、ステップ(e)は、実行時コードの実行に基づいたパラメータを作業領域に記憶された通信パターンに関するパラメータと一致するかどうかを比較するようにしてもよい。

【００２５】ステップ(e)は、実行時コードの実行に基づいたパラメータを作業領域に記憶された通信パターンに関するパラメータと一致するかどうかを比較し、パラメータと一致する場合には、さらにメモリアクセスパターンに関するパラメータと一致するかどうかを比較するようにしてもよい。

1次元目 a=1,b=0
1次元目 a=1,b=1

【００２６】ステップ(a)において抽出されるパラメータは、プログラム中のループに関するパラメータ、ループ内の代入文中の左辺及び右辺の配列を分割するプロセッサの個数、配列の各次元の大きさ、配列の各次元の分割方法、配列参照を決定する式にパラメータを有するようにしてもよい。

【００２７】また、上記の通信パターンは、一のプロセッサから他のプロセッサへデータを受け渡しをする領域を記述したものであってもよい。

【００２８】さらに上記のメモリアクセスパターンは、通信パターンに従いデータを受け渡しする際に、あるプロセッサが管理するメモリ領域を記述したものであってもよい。

【００２９】

【実施例】図１は、本発明の実施例を示すフローチャートである。このフローチャートは、コンパイラによる解析とコード生成（ステップ１１からステップ１３）及び実行時コードによる解析と通信パターンの保存（ステップ１４からステップ１８）とから構成されている。

【００３０】キャッシュ解析（ステップ１１）

まず、対象とするループネスト内の代入文を実行する際に発生する通信パターンが、キャッシュ可能かどうか、すなわち、過去の履歴を使用することが可能かどうかを解析する。そのためには、以下のパラメータについて調べる必要がある。

【００３１】(1) ループインデックスの上下限
(2) 左辺の配列の各次元を分割するプロセッサ個数
(3) 左辺の配列の各次元の大きさ
(4) 左辺の配列の各次元の分割方法
(5) 左辺の配列参照を決定する式のパラメータ
(6) 右辺の配列の各次元を分割するプロセッサ個数
(7) 右辺の配列の各次元の大きさ
(8) 右辺の配列の各次元の分割方法
(9) 右辺の配列参照を決定する式のパラメータ

【００３２】キャッシュ可能であるためには、これらのパラメータが以下の条件を満たすことが必要である。

条件１：それぞれのパラメータが、定数又はループネストを実行している間に値が変化しないループ内不変変数であること

この条件を満たす場合には、通信パターンのキャッシュが可能であると判断し、次のステップに進む。

【００３３】キャッシュのためのデータ抽出（ステップ１２）

上記のパラメータのうちで、実行時コードの実行時に変化する可能性のあるパラメータを調べる。そして、通信パターンの履歴またはメモリアクセスパターンの履歴を記憶するために作業領域を設け、変化する可能性があるパラメータの個数だけ、コンパイル時に旧パラメータと現パラメータの領域を作業領域中に確保する。

【００３４】図２は、本実施例における作業領域を示す図である。ここで、図中の「フラグ」には、旧パラメータの値が有効であるならば１を設定し、無効ならば０を設定する。また領域を確保する際には０を設定しておく。また、「通信パターンへのポインタ」は、自プロセッサからデータを一括して送ることのできる配列領域を全てのプロセッサに対して求めた結果としてのポインタを示が示されている。「アクセスパターンへのポインタ」は、プロセッサ間通信パターンに従ってデータを送る際に、自プロセッサのメモリから読み出すベースアドレス、ストライド、要素雄へのポインタが示されている。さらに、「旧パラメータ」には、前回通信パターンを計算した際のパラメータの値を、「現パラメータ」には、現在のパラメータの値が示されている。

【００３５】実行時に変化する可能性のあるパラメータについて、上記の作業領域の現パラメータに格納するコードを生成する。但し、実行時に変化する可能性のあるパラメータであっても、以下の２つの条件を満たすならば、そのパラメータを作業領域に保存するコードを生成しない。

【００３６】条件２：

(1) 配列のある次元をアクセスする式に、ループインデックスが現れない。

(2) 配列のある次元を分割するプロセッサ数が１であるか、又は、プロセッサへ分割されていない。

【００３７】これは、プロセッサ間通信パターンを計算するパラメータの値が異なっても、プロセッサ内のメモリアクセスパターンが異なるだけで、プロセッサ間通信パターンは同じである点に着目したものである。従って、このような条件を満たせば、通信パターンの計算を省略することができる。

【００３８】なお、このとき、配列の次元に関する全てのパラメータを、キャッシュ可能解析のパラメータとして、作業領域に保存することはしない。但し、このパラメータは、実行時に各プロセッサにおいて、プロセッサ間通信を行うデータの読み込み・書き込みを行うメモリアドレスを決定する際のパラメータとして用いる。

【００３９】以上のステップ１２及びステップ１３により、プロセッサ間の通信パターンを決定する複数のパラメータが抽出される。

【００４０】実行時キャッシュ解析のためのコード生成（ステップ１３）

次のステップでの処理を行うためのサブルーチンコールを生成する。

【００４１】キャッシュ可能解析（ステップ１４）

ステップ１３において生成されたサブルーチンコール中のライブラリの処理について以下に説明する。なお、以下の説明は、このライブラリが実行される時点で、コンパイル時に確保した作業領域の現パラメータに、最新の値が入っていることを前提としている。まず、前回利用した通信パラメータが利用可能であるかどうかを調べる（ステップ１４）。すなわち、作業領域における現パラメータと旧パラメータとが一致するかどうかを調べる。なお、このステップについては、後述する。

【００４２】通信データのメモリアクセスパターン計算チェック（ステップ１５）

プロセッサ間通信パターンに従って転送するデータを、メモリから読み出し、又は書き込みするアクセスパターンがキャッシュされたデータと同じであるかどうかをチェックする。ステップ１２における条件２を満たすパラメータが存在しない場合、すなわち、メモリアクセスパターンが変化しない場合には、ステップ１８に進む。条件２を満たすパラメータがある場合には、通信パターンが同じであっても、メモリアクセスパターンが異なる場合があるので、ステップ１７に進め、メモリアクセスパターンを計算する。ステップ１７のメモリアクセスパターンの計算は、次のステップ１６のプロセッサ間通信パターンの計算に比べて、オーバーヘッドが短いので、再計算を実行してもキャッシュの効果は大きい。

【００４３】プロセッサ間通信パターン計算（ステップ１６）

プロセッサ間通信パターンを計算する場合には、以下のどちらかの場合がある。

(1) はじめてプロセッサ間の通信パターンを計算する場合

(2) 前回のプロセッサ間通信パターンを計算したときに用いたパラメータと今回のパラメータとが等しくない場合

【００４４】作業領域の現パラメータの値を用いて、プロセッサ間通信パターンを計算する。通信パターンとは、自プロセッサからデータを一括して送ることのできる配列領域を、全てのプロセッサに対して求めた結果である。

【００４５】通信パターンの計算は以下の手順で行う。

(1) 代入文の左辺の配列の分散から、自プロセッサがまわるループのインデックス範囲を計算する。

(2) (1)で求められたループのインデックス範囲によって右辺の配列について読み込みが行われる領域を計算する。

(3) (2)で求められた配列の読み込みが行われる領域と実際に自プロセッサでもつ領域の差が、プロセッサ間通信パターンとなる。

また、プロセッサ間通信パターンへのポインタをコンパイル時に確保した作業領域に保存する。

9

【００４６】通信データのメモリアクセスパターン計算（ステップ１７）

プロセッサ間通信パターンに従って転送するデータを、メモリから読み出し、または書き込みするアクセスパターンを決定する。メモリアクセスパターンとは、プロセッサ間通信パターンに従って、データを送る際に、自プロセッサのメモリから読み出す、ベースアドレス、ストライド、要素数のことである。また、計算されたメモリアクセスパターンへのポインタを、コンパイル時に確保された作業領域に保存する。

【００４７】プロセッサ間通信の実行（ステップ１８）

キャッシュされた、または再計算されたプロセッサ間通信パターンとメモリアクセスパターンに従って、実際のプロセッサ間通信を行う。

【００４８】ここで、ステップ１４のキャッシュ可能解析についてさらに詳述する。このステップは、作業領域における現パラメータと旧パラメータとが一致するかどうかを調べるステップであるが、これは図３のようなフローを実行し、キャッシュフラグの状態で判断される。

【００４９】まず、キャッシュフラグに１を設定し（ステップ３１）、作業領域からパラメータを１つ選ぶ（ステップ３２）。そして、フラグを１かどうか、すなわち旧パラメータが有効かどうかを判断する（ステップ３２）。フラグが１の場合、すなわち旧パラメータが有効な場合には、現パラメータが旧パラメータと一致しているかどうかを判断する（ステップ３４）。ステップ３３で、フラグが１でない場合、またはステップ３４で現パラメータと旧パラメータとが一致しない場合には、ステップ３５に進み、キャッシュフラグに０をたてると共に、現パラメータを旧パラメータに代入する。ステップ３４がＹｅｓの場合またはステップ３５の処理の後、全てのパラメータを処理したかどうかを判断する（ステップ３６）。Ｙｅｓの場合には、フラグに１をたてた（ステップ３７）後に処理を終了する。Ｎｏの場合には、ステップ３２に戻る。

【００５０】この処理が終了したときにキャッシュフラグが１のままであれば、前回計算したプロセッサ通信パターンと同じものを使用することができる。この場合には、ステップ１５に進む。また、キャッシュフラグが０になった場合には、プロセッサ間通信パターンを再計算する必要があるのでステップ１６に進む。

【００５１】以上の手順により、既に計算された通信パターンを繰り返す場合には、プロセッサ間の通信パターンの再計算を省略することができる。次に、実際のプログラムを例に本発明を説明する。

【００５２】プロセッサ間通信パターンの計算を省略する方法

下記のプログラムリスト２は、差分法の一例であり、実行するプロセッサは４台で、配列Ａを分割している。

【００５３】

10

【数５】

[プログラムリスト２]

```
      SUBROUTINE SUB(A,N)

      REAL A(N)
!HPF$ PROCESSORS P(4)
!HPF$ DISTRIBUTE (BLOCK) onto P :: A
      DO TIME=1,10
        DO I=2,N-1
          A(I)=A(I-1)+A(I)+A(I+1)
        ENDDO
      ENDDO
      END
```

【００５４】このプログラムを本発明を適用してコンパイルすると、以下のようなコードが生成される。

【００５５】

【数６】

```
      SUBROUTINE SUB(A,N)
      REAL A(N)

      DO TIME=1,10
       w0.array_ub(1)=N
       w0.loop ub(1)=N-1
       call Compute_LIS(A(I), LIS, w0)

       w1.array_ub(1)=N
       call Do_Prefetch_Comm(LIS, A(I+1), w1)
       w2.array_ub(1)=N
       call Do_Pipeline_Recv(LIS, A(I-1), w2)

       DO I=LIS.LB(1),LIS.UB(1)
         A(I) =A(I-1)+A(I)+A(I+1)

       ENDDO

       w3,array_ub(1)=N
       call Do_Pipeline_Send(LIS, A((I-1), w2)

      ENDDO
```

【００５６】以下では、N=100と仮定して１番プロセッサの振る舞いについて説明するが、２番から４番プロセッサについても同様である。まず、TIME=1の場合を考える。Compute_LISで、１番プロセッサが計算を実行するループインデックスＩの範囲を計算する。このとき、上述の(1)乃至(9)のパラメータのうち、(1)乃至(5)のパラメータが必要になる。

【００５７】

【数７】

(1) ループインデックスの上下限：　2,N-1
(2) 左辺の配列の各次元を分割するプロセッサ個数：1次元目 4
(3) 左辺配列の各次元の大きさ：　1次元目 N
(4) 左辺の配列の各次元の分割方法：　1次元目 BLOCK
(5) 左辺の配列参照を決定する式のﾊﾟﾗﾒｰﾀ(aI+b)：　1次元目 a=1,b=0

【００５８】実行時に変化する可能性のあるパラメータは、配列の大きさNと、ループインデックスの上限のN-1であるから、この２つをコンパイル時に確保した作業領域w0の旧パラメータに保存して、Compute_LISを呼ぶ。初めて呼ばれたときには、作業領域内のフラグが０になっているので、フラグを１にして、現パラメータを旧パラメータ領域にコピーした後に、インデックスの範囲を計算する。これらのパラメータから、プロセッサ１が担当するインデックスの範囲を計算し、Fortran90の三つ組み形式で表現すると[2:25:1]となる。

【００５９】プロセッサ間通信パターンを計算するためには、このインデックス範囲と、上述の(1)乃至(9)のパラメータのうち、(6)乃至(9)のパラメータが必要になる。

【００６０】
【数８】

(6) 右辺の配列の各次元を分割するプロセッサ個数：1次元目 4
(7) 右辺配列の各次元の大きさ：　1次元目 N
(8) 右辺の配列の各次元の分割方法：　1次元目 BLOCK
(9) 右辺の配列参照を決定する式のﾊﾟﾗﾒｰﾀ(aI+b)：　1次元目 a=1,b=-1
　1次元目 a=1,b=0
　1次元目 a=1,b=1

【００６１】実行時に変化する可能性のあるパラメータは、配列の大きさのNであるから、これをコンパイル時に確保した作業領域w1,w2,w3の旧パラメータに保存して、通信計算のためのライブラリを呼ぶ。このループ内の代入文の右辺に現れる配列は３つあるので、これらのパラメータから、それぞれの右辺の配列参照についてプロセッサ間通信パターンを計算すると、結果は次のようになる。

【００６２】A(I-1)に関する通信は、[26:26:1]をプロセッサ２からループ実行直後に送信。A(I)は、左辺と同じアクセスなので通信は不要。A(I+1)に関する通信は、[26:26:1]をプロセッサ２へループ実行直前に受信。

【００６３】従って、A(I+1)に関するパラメータを作業領域w1に保存して、ループ実行直前にデータの送受信を行う。さらにA(I-1)に関するパラメータを作業領域w2,w3に保存して、ループ実行直前にデータを受信し、ループ実行直後にデータの送信を行う。

【００６４】次に、TIME=2の場合を考える。TIME=2の場合においても、TIME=1の場合と同様に１００である。まず、Compute_LISでループインデックスの範囲を計算する。Compute_LISを呼ぶ際に、配列の大きさNと、ループインデックスの上限N-1を、作業領域w0の旧パラメータに保存してから呼ぶ。今回は、作業領域内のフラグが１なので、TIME=1のときの旧パラメータと今回の現パラメータとを比較する。その結果、全てのパラメータが等しいので,TIME=1の時に使用したループインデックス[2:25:1]を再使用する。

【００６５】その後、プロセッサ間通信パターンを計算する。まず、A(I+1)に関する通信について考える。Do_Prefetch_Commを呼ぶ際に、配列の大きさNを作業領域w1の旧パラメータに保存してから呼ぶ。今回は、作業領域内のフラグが１であるから、TIME=1のときの旧パラメータと今回の現パラメータとを比較する。この結果、全てのパラメータが等しいので、TIME=1の時に使用した[26:26:1]をプロセッサ２へループ実行直後に送る、という通信パターンを再使用する。以後、TIME=3からTIME=10までTIME=1のときの通信パターンを再使用できるので、通信時のオーバーヘッドを最小限にし、高速なプロセッサ間通信が可能となる。

【００６６】<u>プロセッサ内のメモリアクセスパターンが異なるだけで、通信パターンを省略する方法</u>

下記のプログラム３は、差分法の一例である。実行するプログラムは４台で、配列Ａの１次元目を分割している。配列Ａの２次元目は分割されていない。

【００６７】
【数９】

[プログラムリスト３]

```
      SUBROUTINE SUB(A,N)
      REAL A(N, 2)
!HPF$ PROCESSORS P(4)
!HPF$ DISTRIBUTE (BLOCK, *) onto P :: A
      K=1
      DO TIME=1,10
        DO I=2,N-1
          A(I,3-K) = A(I-1,K)+A(I,K)+A(I+1,K)
        ENDDO

        K=3-K
      ENDDO
      END
```

【００６８】このプログラムを本発明を適用してコンパイルすると、以下のようなコードが生成される。

13

【００６９】
【数１０】

```
SUBROUTINE SUB(A,N)
REAL A(N)

K=1
DO TIME=1,10
 w0.array_ub(1)=N
 w0.loop_ub(1)=N-1
 call Compute_LIS(A(I), LIS, w0)

 w1.array_ub(1)=N
 call Do_Prefetch_Comm(LIS, A(I+1), w1, K)
 w2.array_ub(1)=N
 call Do_Pipeline_Recv(LIS, A(I-1), w2, K)

 DO I=LIS.LB(1),LIS.UB(1)
   A(I,K) =A(I-1,K)+A(I,K)+A(I+1,K)
 ENDDO

 w3,array_ub(1)=N
 call Do_Pipeline_Send(LIS, A((I-1), w2, K)


 K=3-K
ENDDO
```

【００７０】以下では、N=100と仮定して１番プロセッ

(1) ループインデックスの上下限：　2,N-1

(2) 左辺の配列の各次元を分割するプロセッサ個数：1次元目 4
　　2次元目 0

(3) 左辺配列の各次元の大きさ：　1次元目 N
　　2次元目 N

(4) 左辺の配列の各次元の分割方法：　1次元目 BLOCK
　　2次元目 *

(5) 左辺の配列参照を決定する式のﾊﾟﾗﾒｰﾀ(aI+b)：　1次元目 a=1,b=0
　　2次元目 a=0,b=K

【００７２】ループインデックスの決定に影響し、実行時に変化する可能性があるパラメータは、配列の１次元目の大きさNと、ループインデックスの上限N-1であるから、この２つをコンパイル時に確保した作業領域w0の旧パラメータに保存して、Compute_LISを呼ぶ。

【００７３】初めて呼ばれた場合には、作業領域内のフラグが０になっているので、フラグを１にして、現パラメータを旧パラメータ領域にコピーした後に、インデックスの範囲を計算する。これらのパラメータから、プロセッサ１が担当するインデックスの範囲を計算し、Fortran90の三つ組み形式で表現すると[2:25:1]となる。

【００７４】プロセッサ間通信パターンを計算するためには、このインデックス範囲と、上述の(1)乃至(9)のパラメータのうち、(6)乃至(9)のパラメータが必要になる。

【００７５】
【数１２】

(6) 右辺の配列の各次元を分割するプロセッサ個数；1次元目 4
　　2次元目 0

(7) 右辺配列の各次元の大きさ：　1次元目 N
　　2次元目 N

(8) 右辺の配列の各次元の分割方法：　1次元目 BLOCK

14

サの振る舞いについて説明するが、２番から４番プロセッサについても同様である。まず、TIME=1の場合を考える。COMPUTE_LISで、１番プロセッサが計算を実行するループインデックスＩの範囲を計算する。このとき、上述の(1)乃至(9)のパラメータのうち、(1)乃至(5)のパラメータが必要になる。

【００７１】
【数１１】

2次元目 *

(9) 右辺の配列参照を決定する式のﾊﾟﾗﾒｰﾀ(aI+b)：　1次元目 a=1,b=-1
2次元目 a=0,b=K
1次元目 a=1,b=0
2次元目 a=0,b=K
1次元目 a=1,b=1
2次元目 a=0,b=K

【００７６】プロセッサ間通信パターンに影響があるパラメータは、配列がBLOCKで分割されている１次元目のパラメータだけで、さらに実行時に変化する可能性があるのは、配列の大きさのNだけであるから、これをコンパイル時に確保した作業領域w1,w2の旧パラメータに保存する。また、配列が分割されていない次元に現れる配列参照式のループインデックスでない変数Kについては、メモリアクセスパターンを計算するため実行時ライブラリにパラメータを渡して、通信計算のためのライブラリを呼ぶ。このループ内の代入文の右辺に現れる配列は３つあるので、それぞれについて計算する。作業領域に保存されたパラメータから、それぞれの右辺の配列参照についてプロセッサ間通信パターンを計算し、以下の結果を得る。

【００７７】A(I-1,K)に関する通信は、[26:26:1]をプロセッサ２へループ実行直後に送信。A(I,K)は、左辺と同じアクセスなので通信は不要。A(I+1,K)に関する通信は、[26:26:1]をプロセッサ２からループ実行直前に受信。

【００７８】従って、A(I+1)に関するパラメータを作業領域w1に保存して、ループ実行直前にデータの送受信を行う。さらにA(I-1)に関するパラメータを作業領域w2に保存して、ループ実行直前にデータを受信し、ループ実行直後にデータの送信を行う。

【００７９】次に、TIME=2の場合を考える。TIME=2の場合においても、TIME=1の場合と同様に１００である。但し、Kは２になっている。まず、Compute_LISでループインデックスの範囲を計算する。Compute_LISを呼ぶ際に、配列の大きさNと、ループインデックスの上限N-1を、作業領域w0の旧パラメータに保存してから呼ぶ。今回は、作業領域内のフラグが１なので、TIME=1のときの旧パラメータと今回の現パラメータとを比較する。その結果、全てのパラメータが等しいので,TIME=1の時に使用したループインデックス[2:25:1]を再使用する。

【００８０】その後、プロセッサ間通信パターンを計算する。まず、A(I+1)に関する通信について考える。Do_Prefetch_Commを呼ぶ際に、配列の大きさNを作業領域w1の旧パラメータに保存してから呼ぶ。今回は、作業領域内のフラグが１であるから、TIME=1のときの旧パラメータと今回の現パラメータとを比較する。この結果、全てのパラメータが等しいので、TIME=1の時に使用した[26:26:1]をプロセッサ２へループ実行直後に送る、という通信パターンを再使用する。A(I-1)に関する通信も同様にして、[26:26:1]をプロセッサ２からループ実行直前に受け取る、という通信パターンを再使用する。但し、メモリアクセスパターンについては、前回と異なる可能性があるので再計算する。パラメータKを使って計算すると、K=2であるので、以下のようになる。

【００８１】A(I-1)に関しては、[26:26:1][2]を通信データとして読み込む。A(I+1)に関しては、[26:26:1][2]へ受け取ったデータを書き込む。

【００８２】以後、TIME=3からTIME=10までTIME=1のときの通信パターンを再使用し、メモリアクセスパターンを再計算するだけでよいので、通信時のオーバーヘッドを最小限にし、高速なプロセッサ間通信が可能となる。

【００８３】

【効果】従来の分散メモリ型並列コンピュータでは、過去の通信履歴を利用してプロセッサ間通信パターンの計算を省略することができるので、プロセッサ間の通信を高速化することができる。

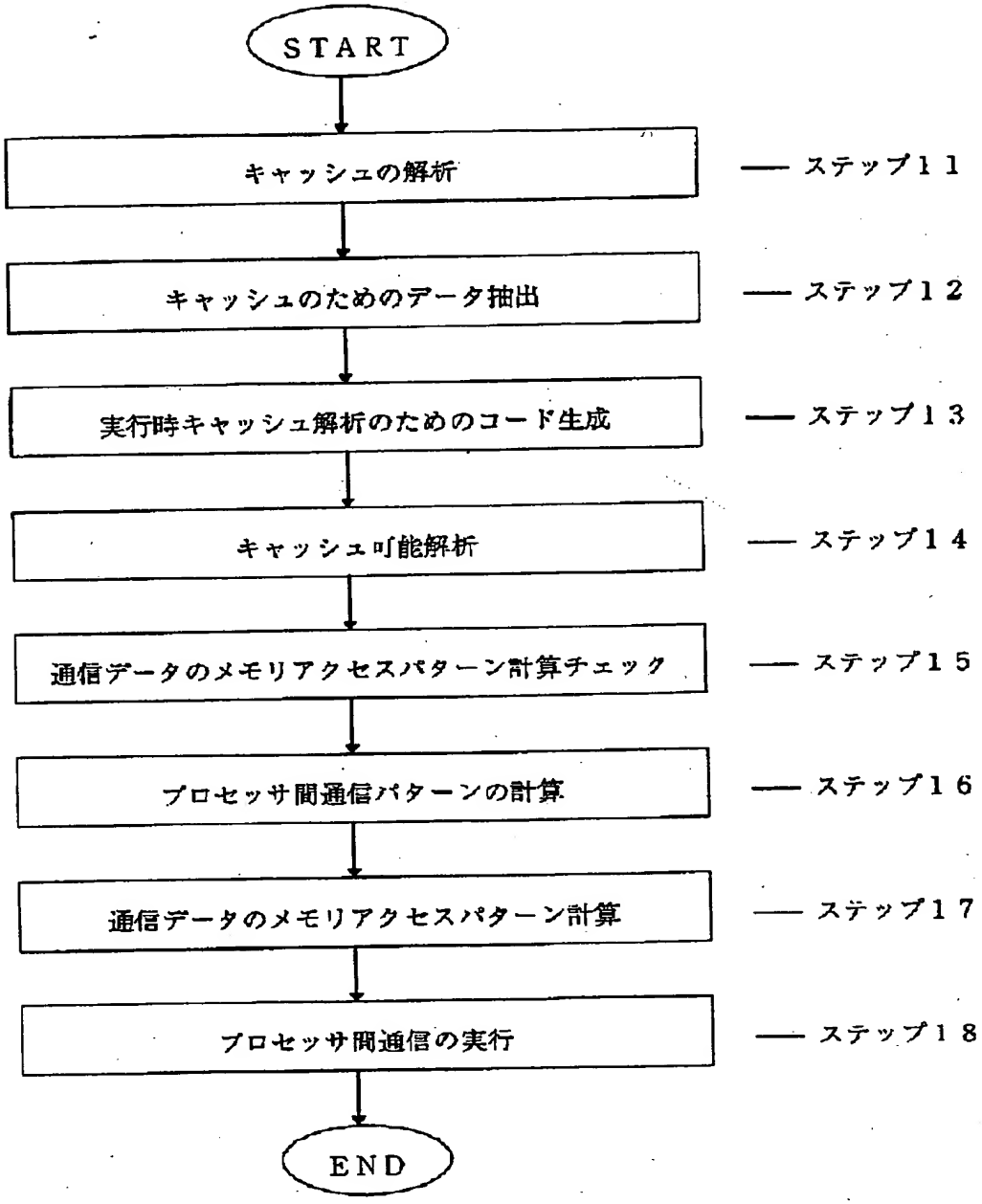
【図面の簡単な説明】

【図１】本発明の実施例を示すフローチャートである。

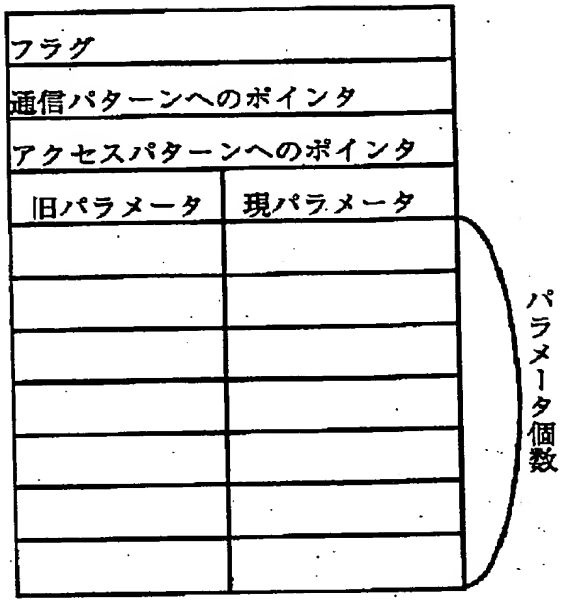
【図２】本実施例における作業領域を示す図である。

【図３】キャッシュ可能解析について詳述したフローチャートである。

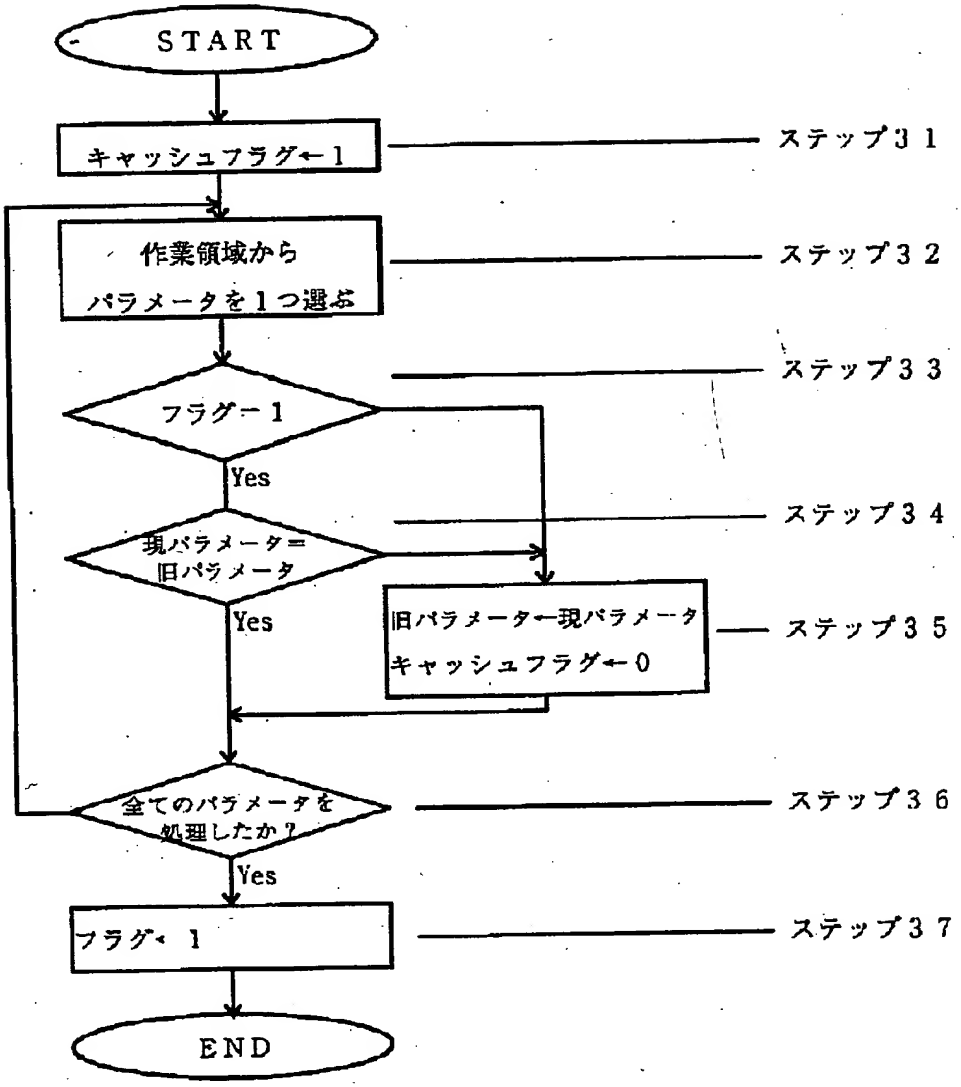
【図１】


START
キャッシュの解析
ステップ１１
キャッシュのためのデータ抽出
ステップ１２
実行時キャッシュ解析のためのコード生成
ステップ１３
キャッシュ可能解析
ステップ１４
通信データのメモリアクセスパターン計算チェック
ステップ１５
プロセッサ間通信パターンの計算
ステップ１６
通信データのメモリアクセスパターン計算
ステップ１７
プロセッサ間通信の実行
ステップ１８
END

【図２】



【図３】


START
キャッシュフラグ←１
ステップ３１
作業領域から
パラメータを１つ選ぶ
ステップ３２
フラグ＝１
ステップ３３
Yes
現パラメータ＝
旧パラメータ
ステップ３４
Yes
旧パラメータ←現パラメータ
キャッシュフラグ←０
ステップ３５
全てのパラメータを
処理したか？
ステップ３６
Yes
フラグ←１
ステップ３７
END

フロントページの続き

(72)発明者　小笠原　武史
神奈川県大和市下鶴間１６２３番地１４
日本アイ・ビー・エム株式会社東京基礎研
究所内